[0152] The manifest is used to create a prototype. The act of installing a manifest and its described artifacts into a system creates a prototype. A prototype is a "runable" or "executable" manifestation of a software component. The prototype is used to create an instance or an abstraction of a software component. An instance or abstraction is a "running" or "executing" manifestation of a prototype.

[0153] In the example structure **500**, the right-to-left arrows are to be read as "is described by." So, for example, the top tier **510** may read this way: OS abstraction **512**"is described by" its system prototype **514**, which "is described by" its system manifest **516**.

[0154] In this structure, the left-to-right arrows are to be read as "is used to create." So, for example, the second tier **520** may read this way: Application manifest **526**"is used to create" its application prototype **524**, which "is used to create" its application abstraction **516**.

[0155] Similarly, the members of each tier have a defined relationship between members in other tiers. Each member of a progressively higher tier references or includes the like members of lower tiers.

[0156] In the example structure **500**, the top-to-bottom arrows are to be read as "includes or "supervises." So, for example, the system manifest **516**"includes" the application manifest **526**, which "includes" the process manifest **536**, which "includes" the load-source manifest **546**.

[0157] In this structure, the bottom-to-top arrows are to be read as "contained in" or "is supervised by." So, for example, the process prototype **534**"is supervised by" the application prototype **524**, which "is supervised by" the systems prototype **514**.

[0158] In alternative implementations, the exemplary self-describing artifact architecture has a different number of tiers, different arrangements of tiers, and/or different abstractions.

[0159] The systems manifest **516** describes the entire system (i.e., all of the artifacts). It is a top-level manifest pointing to manifests for each operating system component and each application. Depending on scope, individual operating system components are described with application, process, or load source manifests.

[0160] The application manifest **526** contains the process manifest **536**. As such, the application manifest describes or specifies the processes that are created when the application (represented by application abstraction **522**) runs. Application manifests may also identify the interprocess communication interfaces exposed or required by the application and describe the bindings between interprocess communication interfaces of processes within the application.

[0161] The process manifest **526** contains load module manifests describing or specifying the load modules included in the process (represented by the process abstraction **532**). Process manifests may identify the interprocess communication interfaces exposed or required by each process and describe the bindings between code and data interfaces on load modules.

[0162] The load source manifest **536** describes or specifies the persisted binary file containing the executable code of the load module and identifies any further load modules required by this load module. Load source manifests identify the code and data interfaces exposed or required by the load module.

[0163] An embodiment may support several types of manifests including manifests for running processes and for process prototypes, manifests for running applications and for application prototypes, manifests for running operating system components and for their prototypes, manifests for hardware devices, and one or more manifests for the system as a whole. In such an embodiment, the differing manifests may share and reuse the same structural elements.

Application Abstraction

[0164] Rather then just being part of a user-centric model, the concept of an application program is actually part of this the exemplary self-describing artifact architecture **100**. In particular, the OS **112** (or portions thereof) inherently recognize the concept of an "application abstraction." As described by its associated manifest, an application abstraction is descriptively and necessarily linked to specific low-level abstractions (such as active processes and their load sources) and specific high-level abstractions (such as the operating system).

[0165] When a user, either directly or indirectly, runs a program, the OS **112** creates an instance of an application abstraction (such as application abstraction **522**) from an application prototype (such as application prototype **524**). Creating an instance of an application includes creating new instances of processes described by their process prototypes.

[0166] The static description of the application is embodied in one or more application manifests (such as application manifest **526**). The OS **112** maintains "dynamic" metadata that links processes with applications, processes with process prototypes, and applications with application prototypes. The OS **112** also maintains additional dynamic metadata that links process prototypes and application prototypes with their respective manifests.

[0167] The application abstraction is embodied in a dynamic object. Other software components (such as part of the OS) can communicate with the dynamic application abstraction objects to determine which applications are running, to determine which processes belong to an application, and to retrieve other metadata, such as manifests, that are also available. For example, given the identity of a process, a program can ask the OS for the identity of the application to which it belongs; given the identity of an application, a program can ask the operating system for the identity of its application prototype; etc.

Methodological Implementation of Exemplary Application Abstraction Management

[0168] **FIG. 6** shows a method **600** performed by the OS **112** or portions thereof for the purpose of creating and managing application abstractions. This methodological implementation may be performed in software, hardware, or a combination thereof. For ease of understanding, the method is delineated as separate steps represented as independent blocks in **FIG. 6**; however, these separately delineated steps should not be construed as necessarily order dependent in their performance. Additionally, for discussion purposes, the method **600** is described with reference to **FIG. 1**.